UNCLASSIFIED

AD 296 962

Reproduced
by the

ARMED SERVICES TECHNICAL INFORMATION AGENCY
ARLINGTON HALL STATION
ARLINGTON 12, VIRGINIA

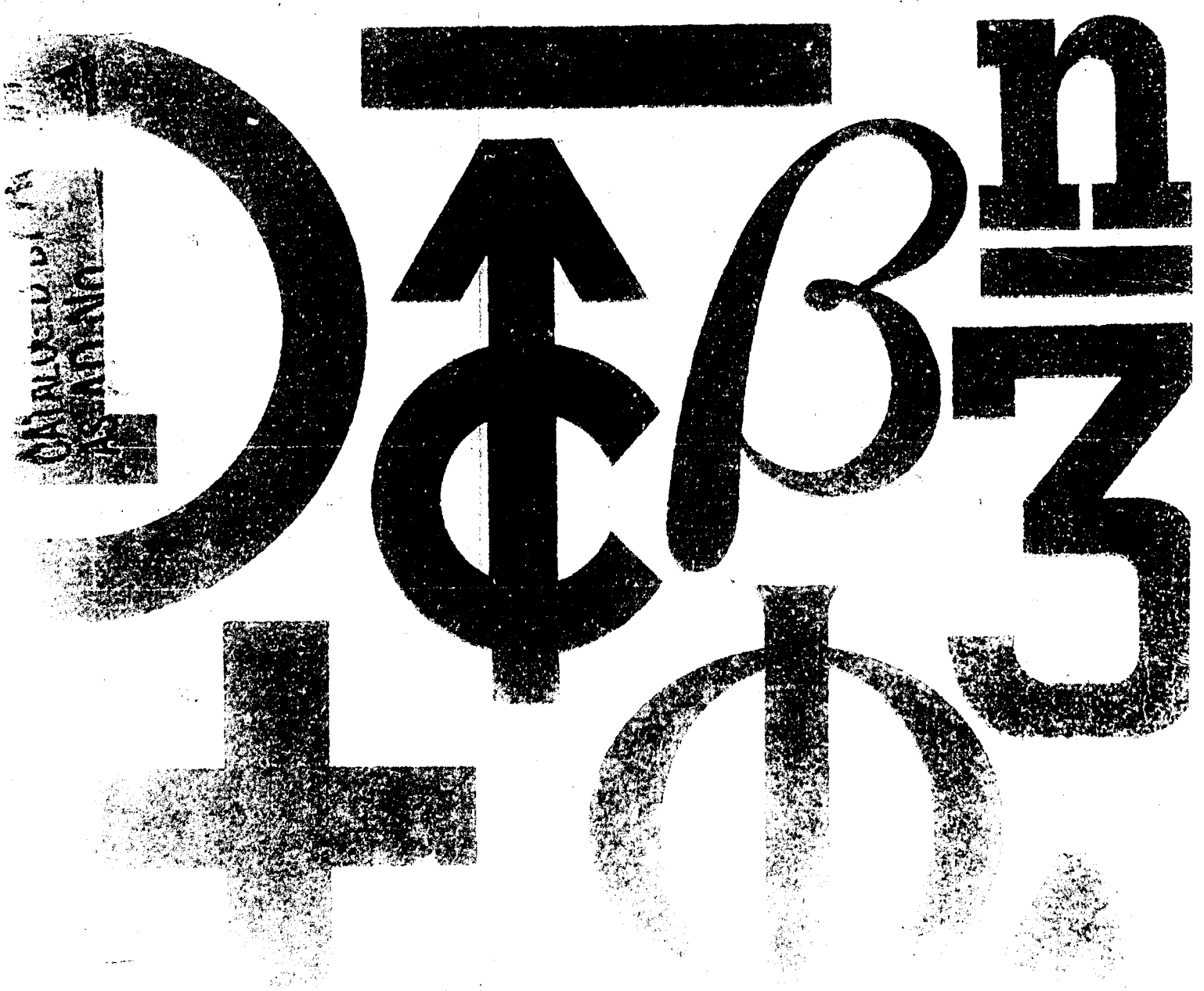
UNCLASSIFIED

NOTICE: When government or other drawings, specifications or other data are used for any purpose other than in connection with a definitely related government procurement operation, the U. S. Government thereby incurs no responsibility, nor any obligation whatsoever; and the fact that the Government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data is not to be regarded by implication or otherwise as in any manner licensing the holder or any other person or corporation, or conveying any rights or permission to manufacture, use or sell any patented invention that may in any way be related thereto.

A PROBLEM-ORIENTED SYMBOL PROCESSOR

IBM RESEARCH

296 962



RC 840

Best Available Copy

AFCRL - 62 - 964

ADAM - A PROBLEM ORIENTED SYMBOL PROCESSOR

Alvin P. Mullery
Ralph F. Schauer

International Business Machines Corporation
Thomas J. Watson Research Center
Yorktown Heights, New York

Rex Rice

Data Systems Division
182 North Hamilton Street
Poughkeepsie, New York

Contract No. AF19(628)-1621
Project No. 4641 Task 464105
Scientific Report No. 1

December 20, 1962

Prepared
for

ELECTRONICS RESEARCH DIRECTORATE
AIR FORCE CAMBRIDGE RESEARCH LABORATORIES
OFFICE OF AEROSPACE RESEARCH
UNITED STATES AIR FORCE
BEDFORD, MASSACHUSETTS

Requests for additional copies by Agencies of the Department of Defense, their contractors, and other government agencies should be directed to the:

ARMED SERVICES TECHNICAL INFORMATION AGENCY
ARLINGTON HALL STATION
ARLINGTON 12, VIRGINIA

All other persons and organizations should apply to the:

U. S. DEPARTMENT OF COMMERCE
OFFICE OF TECHNICAL SERVICES
WASHINGTON 25, D. C.

Scientists and engineers who contributed to the work reported:

R. J. Arculeo

V. DiLonardo

J. M. Gooch

R. A. Howard

A. P. Mullery

P. R. Osseck

A. T. Pfeiffer

R. H. Riekert

R. F. Schauer

J. J. Shea

L. M. Terman

R. Rice

ABSTRACT

A study of the general area of problem solving with a digital computer revealed characteristics of data that are essentially ignored or suppressed in conventional systems. In an attempt to increase the capability and flexibility of a digital system a new high-level language has been defined which utilizes these data characteristics. A machine organization which implements this language as a machine language and yet imposes no restrictions on the use of the language has been proposed. The system will have the following characteristics as a result of this organization:

1. Complete symbolic addressing on variable field length data. This addressing scheme eliminates any consideration of the physical location of the data in the system once the system has ever seen the data.

2. List and string operations. These operations include arithmetic operations as well as those such as deletion, insertion, union, intersection, etc.

3. High to low order numeric processing. This technique permits all data to be handled in exactly the same way.

4. Dynamic storage allocation. The machine assigns all storage and maintains a continuously updated blank or availability list.

5. Automatic input-output. Machine control of the storage of variable field length data and instructions necessitates a procedure for machine control of input-output functions.

INTRODUCTION

Digital computers have evolved in their own technical environment, and to a large degree independently of the problem environment. Thus it was necessary to have computing centers with staffs of programmers as intermediaries between machines and users. As the inadequacy of the arrangement became apparent, problem-oriented languages were written, with compiler programs to allow the machines themselves to do the conversion to their own (machine) language. Accommodating to the nature of the computer in this way still was not the answer from the scientist's or experimenter's point of view, for there remained an enormous commitment of processing (compiling) and debugging prior to the first feedback of results. Furthermore, it proved necessary to write compilers for many problem fields, which gave this mode of solution a patchwork look. For these reasons, we decided to attack the problem at its roots by changing the fundamental nature, i.e., the organization of the computer itself. We took it as the aim of our work to allow the experimenter to use the computer as directly as possible as an experimental tool.

We reasoned that "data" upon which machines operated have certain similar characteristics even though problems in which the data are used vary. We therefore began by making a study of the nature of data. In general, data is not just of the form acceptable by most present day computers--a contiguous string of fixed length, fixed point data, but is variable field length, variable format, structured, and not necessarily contiguous. For example, a three-by-three matrix is a string of nine data symbols. But more than just a string of symbols, a hierarchy or grouping is usually imposed on the string. In this matrix example, the grouping consists of symbols which are contained in rows in a matrix. This is a simple grouping The English language, considered as data, is grouped as words in phrases, in sentences, in paragraphs, in chapters, in books, and in libraries. No matter what one calls these groupings, the structure they indicate does exist in data. Clearly each symbol in a string of data could contain any number of characters.

Further, a string of data need not be contiguous. In text, for example, a footnote is part of a string, linked (by a symbol) but physically removed. It is necessary to have a way of storing non-contiguous data in the computer and of providing linkages. Where such data applies to many strings, it should be stored once and linked as necessary. An analogy is a reference in text cited frequently though given only once. Insert procedures involving the movement of entire strings of data are inefficient and should be unnecessary.

If the data to be processed has variable field length, variable format, structure, and is not necessarily contiguous, any language or notation to describe the data or operations on it must not restrict but take advantage of this form. This leads to the following implications concerning the language and the system:

1. That it be able to handle variable field length data and instructions.
2. That it be able to maintain the structure of the data internally.
3. That it be able to use and operate on this structure.
4. That it be able to handle symbolic addressing of the data.
5. That it be able to interpret links in the data correctly.
6. That it be able to insert links in the data when necessary.

In addition, in order to be as general as possible, the machine system should not impose limits on the number of levels of links, the number of names in the data, the size of the data or instruction fields, etc.

The language must include operators to permit the three basic data operations--creation, movement, and destruction. The methods by which data can be created are limitless. Some, however, are more commonly used than others. These include: Add, Subtract, Multiply, Divide, AND, OR, NOT, and Reproduce. Basic data movement operations include Insert, Separate, and Join. Data destruction is accomplished with a Delete operation. All of these operators must be defined for sets or strings of data as well as single fields. Probably the most important feature of such a high-level language is the ability to easily define new operators. This ability to define operators must be simple and flexible, and the execution of the resulting subroutines must be fast and efficient. There should

be no limit on the number of levels of subroutines; that is, subroutines must be able to contain other subroutines which can contain other subroutines, etc.

The use of the language should reduce the need of a program controlled housekeeping to a minimum. As examples, the assignment of storage space should be completely automatic, and data operations should be independent of data format; that is, if the data is not in the proper format for any given operation, machine control will do the necessary conversion from any permissible format to the desired one

In order to implement this language efficiently, a completely new machine organization has been evolved. This organization has been determined only by the above goals. This has meant new concepts in the organization of the storage, of the process unit, of the input-output, of instruction and data flow paths, of controls, etc.

The following sections describe the characteristics of the data, of the language, and of the resultant machine organization.

DATA

It was indicated in the introduction that data has structure. It may be a simple structure such as rows and symbols for a matrix or it may be the complex structure such as that in the English language. In any case the symbol is the lowest meaningful grouping--e.g., the character "b" has no meaning, but the characters "ball" do represent something and, therefore, make up a symbol. The four characters b-a-l-l are a symbol for the physical object, a ball.

No matter what these groupings may be called, they do exist within the data and are indicated by identifiers. These identifiers are usually special symbols which indicate the end of a group and the start of a new group. For example, a record mark ≢ indicates a boundary between two records. Many different such marks have been used, depending upon the names which have been given to the grouping. In order to avoid unnecessary confusion, the groups

Character
Symbol
Phrase
Sentence
Paragraph
Chapter
Book
Library

shall be used throughout. The identifiers associated with each group shall be as follows:

Group	Identifier	
	Use	Mention
Character	(Implied)	⓪
Symbol	①	①
Phrase	②	②
Sentence	③	③
Paragraph	④	④
Chapter	⑤	⑤
Book	⑥	⑥
Library	⑦	⑦

When the structure of a string of data or instructions is to be identified, then the "USE" identifiers are used. If some group of data is being referred to in the instructions, then the "MENTION" identifiers are used. The ① identifier will indicate the end of a symbol and the start of a new symbol. These identifiers form a hierarchy in that a ② also implies a ①, a ③ implies a ② , and a ① , etc. If more than one identifier ever appear together with no other characters separating them, all but the highest identifier will be neglected and dropped. Each symbol may contain any number of any characters. Each phrase may contain any number of symbols, etc.

The names of data will appear with the data itself. A name can be given to any string of data at any level from Symbol to Library. This string may also contain named groups of data. Thus, named groups within named groups are allowed. The name character n will surround the name when it appears with the data. The first character following the second n will be a "MENTION" identifier which will indicate the level of the data being named. If this mention identifier does not appear with the name, then the name is assumed to name data at a level indicated by the first following "USE" identifier. For example, a data string will appear in the data storage as

n ADAM n ④ n ABEL n ③ n EVE $\overset{s}{p}$ I n ②④ - - - - n EVE $\overset{s}{p}$ II n
② - - - - ③ - - - - ② - - - - ④

Here ADAM will name the complete paragraph; ABEL will name the first sentence in that paragraph; EVE $\overset{s}{p}$ I will name the first phrase in the first sentence; EVE $\overset{s}{p}$ II will name the second phrase in the first sentence. The names themselves may be formed from any combination of any number of the general characters A - Z, a - z, 0 - 9, $_{0-9}$, etc.

Certain operations require numeric data. Numeric data may contain any number of the digits 0 through 9 and may contain at most one of each of the characters +, -, . (decimal point), and exponent. Together these represent a unique number. However, such a number can be expressed in many ways. Examples of permissible data format external to the machine are as follows:

① 19 ①
① -19.76 ①
① ex + 02 + .1976 ①

In such numeric data, the decimal point is assumed to be at the right unless it is written elsewhere. A decimal point can occur anyplace in a number if the number does not also have an exponent. If the number is written with an exponent, then it must be expressed so that the mantissa is greater than or equals 0.1 and less than 1.0--the decimal point must be at the left. The number is assumed to be positive unless otherwise indicated. If, however, the number is written in exponential form with the exponent preceding the mantissa, the sign of the number must be written.

Some operations require binary data. Such data may contain any number of the characters 0 and 1. Each symbol of binary data should begin with the base 2 indicator L.

A data string may be linked to another data string by means of a ⑧ link character. Whenever a ⑧ is met in a string of data, the data indicated by

the name following the ⑧ will, in effect, be considered to be part of the original string. A given string of data can be so linked to many data strings. For example, in the data strings

n A n ④ Brown ① is ① a ⑧ Des ① school ① with ① many ⑧ Des ① students ④

n Des n ② very ① good ②

the fourth and ninth symbols of A are both "very". Thus, the data string called A is, in effect,

n A n ④ Brown ① is ① a ② very ① good ② school ① with ① many ② very ① good ② students ④

PROPERTIES OF THE LANGUAGE

The language will consist of verbs, nouns, and modifiers, and rules for their use. These rules are the syntax of the language. Much of the power of the language depends on the syntax. This syntax should be simple and direct but should allow a great flexibility of use. These rules for use should be generally applicable.

An English-like structure is used. The operators are classified as nouns, verbs, adverbs, and adjectives. A noun with any number of adjectives modifying it will make up a noun phrase. The noun phrase must always indicate data contained within the extent of the name used. A verb and any number of adverbs modifying it will make up a verb phrase. An operation is a combination of at least one verb phrase and one noun phrase which accomplishes some process (for example, A + B is an operation). A sentence will consist of at least one operation. All operations in the sentence are dependent on the result of other operations in the sentence and are syntactically independent of operations in other sentences. In general, the process to be performed and where to place the result, if any, must be specified in a sentence.

Some verbs require only an object. Other verbs require both a subject and an object. Still others require any number of objects. A subject or object can be a noun phrase, an operation, or a group of operations. The subject of a verb is assumed to be the result of all operations which preceded the verb in the sentence. If a parenthetical phrase precedes the verb, then only this phrase is the subject of the verb. Exceptions to this are verbs such as *, /, and AND. Here normal rules of precedance apply. Any number of parenthetical phrases may be used, both with algebraic and non-algebraic operations.

The object of a verb phrase is the following noun phrase only. Again, if a parenthetical phrase follows the verb, then this parenthetical phrase is the object. If several objects are required, each object should be

separated by a comma (separator). Each of these objects may be a noun phrase, an operation, or a group of operations.

All adjectives will follow the modified noun. A given noun, however, may have any number of adjectives modifying it. An adjective is considered to modify not just the noun, but the noun as modified by any adjective preceding the particular adjective. An adverb will precede the modified verb. Again, a verb can have any number of adverbs modifying it.

A name may be given to a sentence or group of sentences at any level. A name may not be given to a part of a sentence in instructions. Named sentences or groups of sentences may be contained within other, larger, named groups of sentences. A name of an instruction or group of instructions must be defined as a verb. For example, an instruction might be written as follows:

v Start v ③ A + B → C ③

A named verb contained within another verb is considered to extend to the end of the highest named verb in which it is contained.

NAMES AND SYMBOLIC ADDRESSING

Names of data and instructions may contain any combination of any number of the general characters. In the machine language all data and instructions are referenced by their names. A character combination in the instructions can be operated upon only if it is interpreted as a literal. An absolute address, direct or indirect, will never appear in a program. The machine itself will assign locations in storage to named sequences of data and instructions. In order to accomplish this, a fixed length table with two characters (16 bits) per entry is provided. The address of a location in this table is derived from a name. This mapping may be simple. For example, if 4096 locations are provided, then the middle six bits of each of the first two characters of a name may be used to obtain a location in the table. Of course, any other mapping technique may be used; a hash address scheme using every character of a name, for example, may be more efficient. All names that are being used and which map to the same location in the table will be contained in a closed, linked loop in the main memory. The address in a location in the map table will be that of the name last used in the corresponding loop. Fig. 1 demonstrates the construction of such a loop. In this example, the first two characters are used to derive a location in the table. Thus, all names starting with a particular pair of characters will be contained within a loop. The "MA" loop containing the names MAD, MATRIX, and MAUD is shown. The name MAD is stored at location yy. With the name MAD, is a link address to the next name in the loop, MATRIX. Two addresses follow the name MAD to indicate the location of the "current" item and beginning of the data specified by MAD. Similarly, at location xx, the name MATRIX is followed by the link zz which is the location of the next item in the loop, MAUD. Following MAUD is a link yy, which closes the loop. In the table location is the address zz, of MAUD, which happened to be the last name used of this loop. In the example, the noun MATRIX is to be found. This name maps to the 'MA" location in the

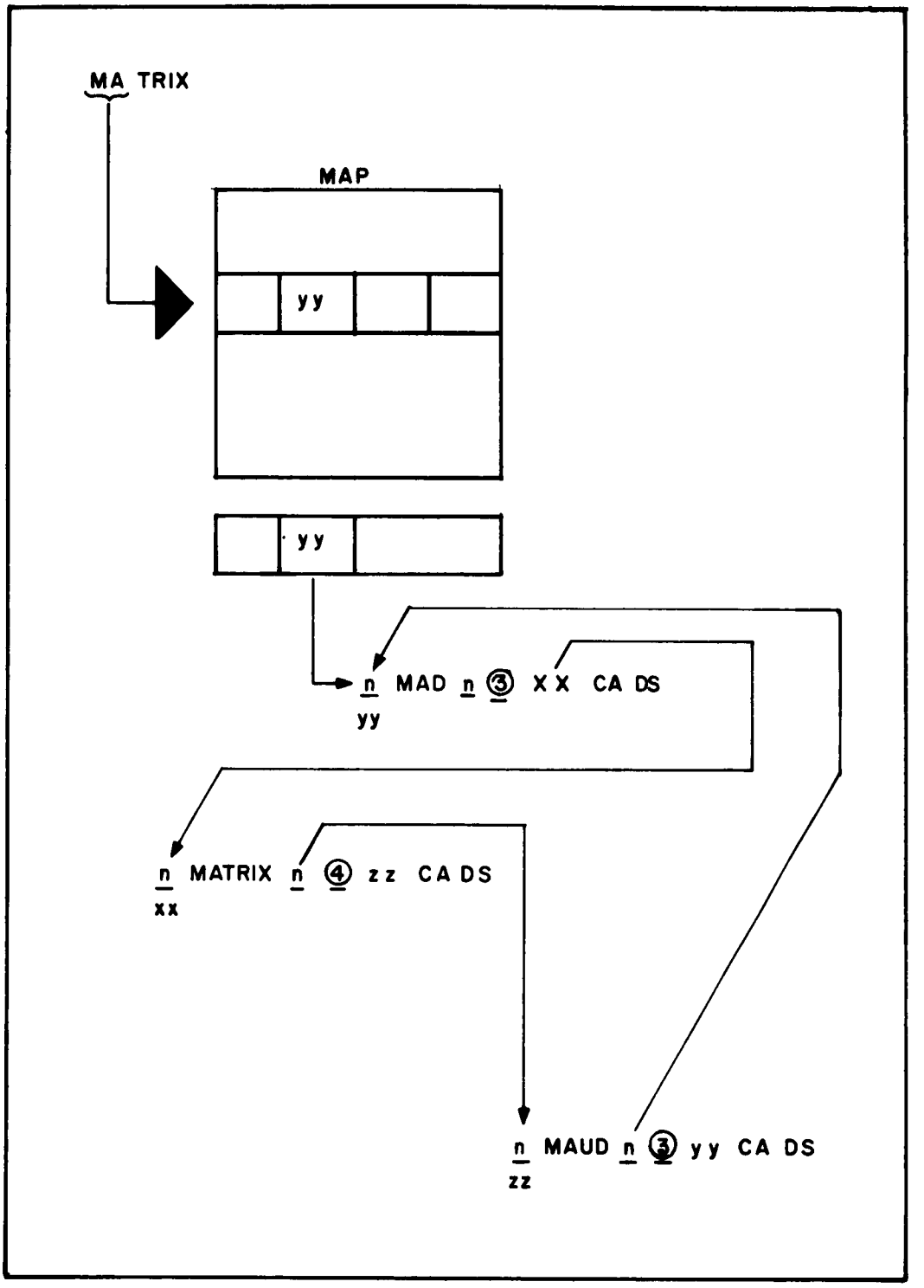


FIG. 1 SYMBOLIC ADDRESSING MAP AND NAME LOOP

table which gives the address xx. At zz, the name MAUD and MATRIX are compared. These are not equal. The name at yy is next compared with MATRIX. Since the comparison again fails, xx is accessed and the names compared. The successful comparison locates data whose name is MATRIX. If a name which is not contained in the loop, MATE, for example, is defined, every name in the loop will be compared with MATE. After going completely through the loop with no successful name comparison, the new name is placed in some available location, say ww, and a new string created. The link address associated with the first item in the loop--here zz with MATRIX--is placed with MATE and the address ww stored with MATRIX. The resulting name loop is shown in Fig. 2. Similarly, when a name is being deleted, the link addresses of the preceding and following names are adjusted to again form a closed loop. The novel feature of this symbolic addressing system is that the size and number of names is limited only by the capacity of main storage. There may be any number of names of any size in a loop. The number of loops is limited by the size of the table and determined by the names in use.

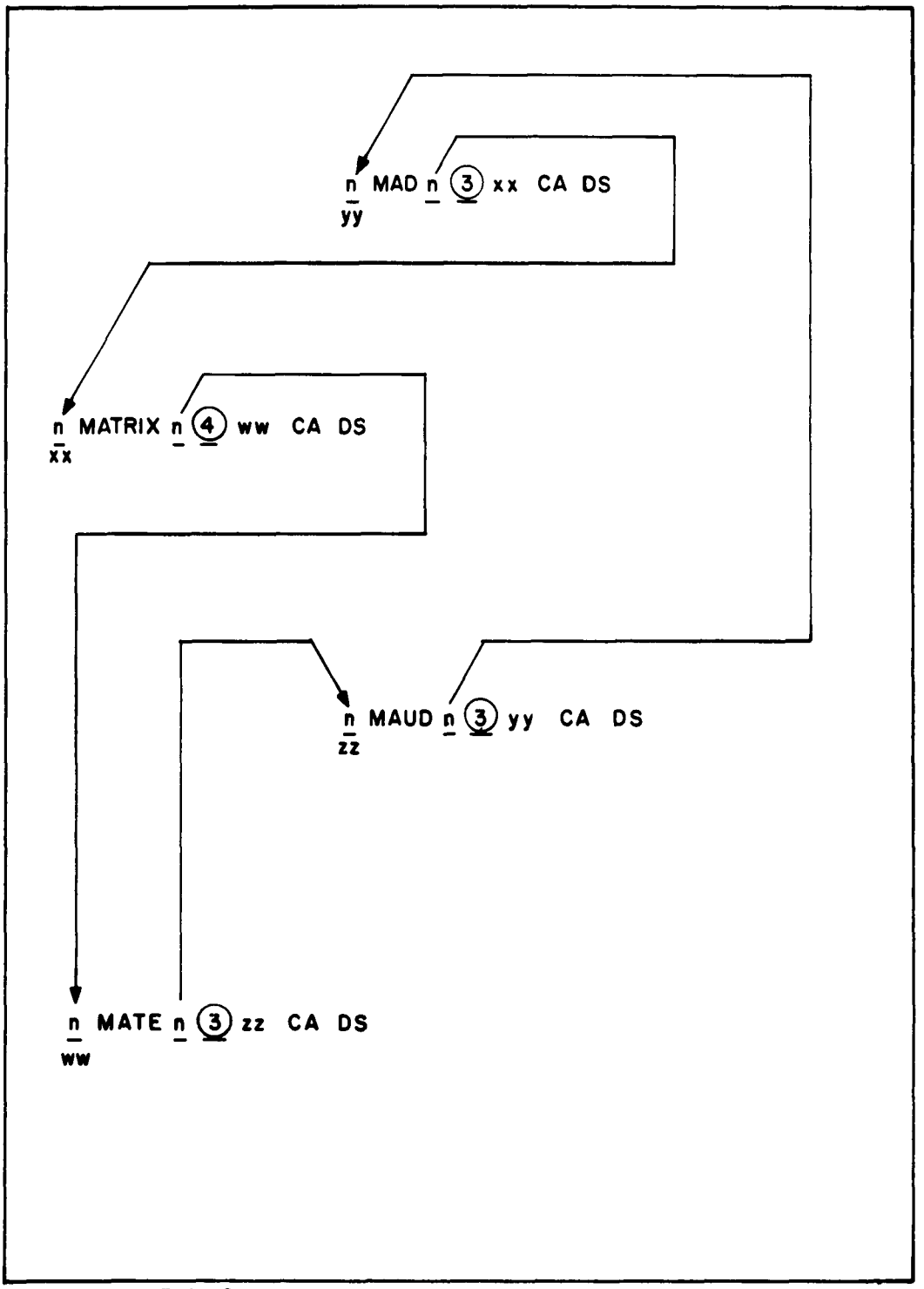


FIG. 2 MODIFIED NAME LOOP

SPECIFICATION OF DATA WITHIN STRUCTURED STRINGS

Particular items within a named data sequence are indicated by means of the $\underline{i}\uparrow j$, $\underline{i}\ \$\ j$, and $\underline{i}\ \cent\ j$ adjectives. In the data string

$$\underline{n}\ A\ \underline{n}\ 3) \text{ --- } 1 \text{ --- } 2 \text{ --- } 1 \text{ --- } 3)$$

A names the whole sentence. The first phrase in this sentence is indicated by the noun phrase $A\ \underline{2}\uparrow 0$. The second symbol of the second phrase is indicated by $A\ \underline{2}\uparrow 1\ \underline{1}\uparrow 1$. In a similar manner, every part of a named string of data can be indicated. Thus, an adjective $\underline{i}\uparrow j$ will indicate the jth item at level $i)$ from the point previously specified. The $\underline{i}\ \$\ j$ adjective has the same effect except that it will also set a "current" indicator at the item referenced. Thus, the noun phrase $A\ \underline{2}\uparrow 1\ \underline{1}\ \$\ 0$ will specify the first symbol of the second phrase of A and mark this symbol current item of the data sequence named A. The adjective $\underline{i}\ \cent\ j$ will find the jth item at the i th level following the "current" item and set this new item as the "current" item. If, for example, succeeding symbols in a string are required, these may be indicated by the noun phrase $A\ \underline{1})\ \cent\ 1$. Each time this noun is interpreted, the next item in A will be indicated without any indexing or other change to the program. In order to perform this type of addressing, the general form memory organization shown in Fig. 3 will be used.

As data or program is input into the machine, it will pass through the input control. The input control will break up the information into machine words for storage in the machine. The beginning of each new classification of data, or every data identifier, will always begin a new machine word. The input control will scan the input for data identifiers and form the variable length block into machine words for storage. The input control will also have partial control over the special control planes associated with the memory. There will be one control plane associated with each data identifier to be used in the system. Whenever an identifier is detected in the input information, a core will be set in the plane identified with this character

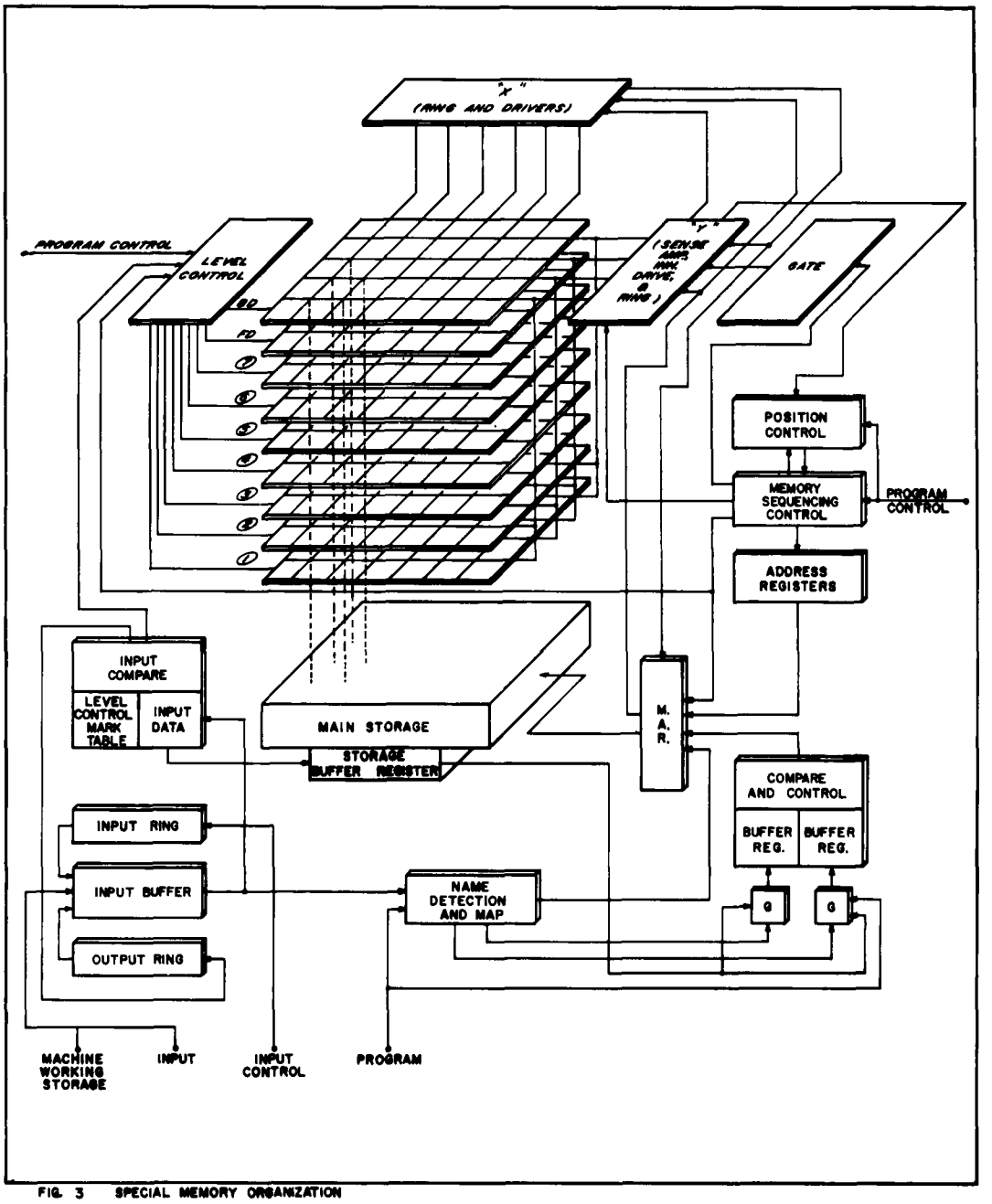


FIG. 3 SPECIAL MEMORY ORGANIZATION

in a position equivalent to the address in which the machine word containing this character is being stored. Similarly, a core will be set in the same position in all lower levels of identifiers explicitly stating what is implied in the data. If an identifier is detected before the storage buffer register is filled, null characters will be inserted to fill out the machine word, and then the word is stored. The identifier will be held by the input control until the storage buffer register is again ready to accept data and will be inserted as the first character of the machine word being formed. When this word is ready for storage, the appropriate related cores will be set in the special control planes. All information will be input in this manner.

The name symbol, verb symbol, and key symbol will also start new machine words when detected by the input control, and marks are set into a special core plane. The use of this mark will be discussed later.

Normally data will be stored in 8-bits and parity outside of the machine whenever possible. This same representation will carry over to internal storage for all alpha-numeric symbols. Certain symbols--those marked as numeric or logic--will be packed for more efficient operation. The input control must detect a numeric field as defined in a previous section and set up the normalization operation. Packing will occur during this latter operation. Logic data will be indicated by a special character. An operation similar to the normalization procedure will be initiated by the detection of this special character by the input control.

With this structure of data stored in the memory, it is possible to find any particular item in a string relatively quickly. Let us say we are looking for

$$A\ \underline{4}\uparrow\ 2\ \underline{3}\uparrow\ 1\ \underline{2}\uparrow\ 2$$

The machine will first find the beginning of the string named A in the manner described previously. Before attempting to describe the search for the specified part of A, consider the organization of the special core planes.

There will be one core plane for each data identifier indicating one level of classification. Each core in this plane will represent one machine word in the main store. The special core planes will be used in a 2-D memory organization so that it will be possible to gate out the contents of 128 cores in one read-write cycle. If a core has been set to a "1", this implies that that level identifier or one higher occurs in the associated machine word. Therefore, by using appropriate circuitry, one is able to scan the contents of a row of one of the special planes from either right or left to determine which of the 128 machine words contain the identifier in question.

The above discussion has assumed that the extra core planes were a separate entity. It is also possible to select a portion of the main store for assembling this information. Even though the main store has a 3-D organization, it is possible to organize it such that the location of identifiers in blocks of machine words (probably 64 at a time) is available in one read-write cycle. This is the organization proposed for the system being designed.

The search for the location of the data of the above example would proceed as follows:

1. Locate the beginning address of the block called "A".
2. In the ④ level plane read out the row in which a location corresponds to this address.
3. Begin the search for ④ marks at this address. As a ④ mark is encountered, count this mark and compare the total with that obtained from the adjective evaluation. Continue the search if the two numbers do not agree.
4. If three ④ marks (magnitude of number in adjective incremented by 1) have not been located and counted by the end of the row, increment the appropriate rings and read out the next row.
5. When three ④ marks have been counted, store the address associated with the last one located. Then read out the ③ level row containing this address.

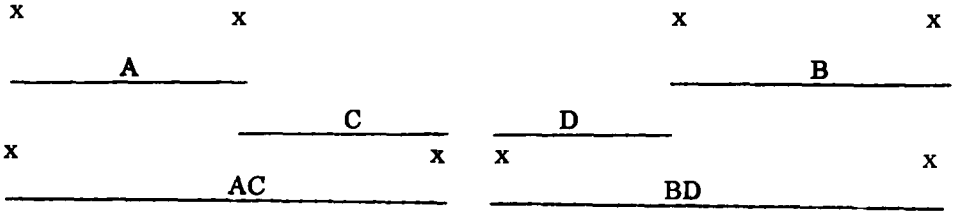
6. Begin searching in the above manner for the second (3) mark starting at the address of the third (4) mark. When this is located, store its address and proceed at the (2) level.
7. When the address is finally located, the machine will look to the instruction control to determine what to do next. It is possible to operate on any information which can be located in the described manner from this newly located point in the data.

The absolute address of the current item in a data string is stored with the name of the string. Thus, when the adjective is (1) j, the scan is begun at this address; and when the described point is located, the current address will be changed.

In order to perform the input procedure described in this section and also to perform some operations such as insert and delete which will be described, the memory organization must be capable of determining and remembering blank memory locations. A special core plane is included to perform this function. The extent of a blank sequence is determined before the location of the sequence is placed in the blank plane.

The special plane will be accessed in exactly the same manner as the other special core planes. In this case a core associated with the address of the beginning of the sequence will be set and one associated with the address immediately following the last address of the sequence will also be set. This technique permits one to combine adjacent blank sequences. This will be accomplished by setting an addressed core in the special plane to a "1" if it previously was a "0" and vice versa.

Consider the following example:



Two blank sequences exist in memory and are labeled A and B. Two additional sequences, C and D, are to be added. When the initial address of C is to be set, the control will set the equivalent core to a "0" because it was previously a "1", indicating the end of A. When the end of C is set, the sequence now contains both A and C. In a like manner, when D is put into the special plane, the end of D removes the beginning of B and gives a single sequence.

A blank sequence will be available to memory control at all times. Information relating to the address of the next blank word and the address of the last word of the sequence is kept by the memory control. When a blank sequence has been filled, the memory control will initiate a search of the special plane starting at the address of the end of the last sequence. The address of the next mark in the special plane will be the address of the beginning of the next blank sequence. The second mark will be the end of the sequence. These addresses will be stored in the memory control. Thus the memory will be loaded in cyclic manner in order to reduce the length of time necessary to locate the next blank sequence.

SEARCH AND STORAGE OF VARIABLE LENGTH DATA

All data has been considered to be of variable length. In addition, the programmer, through such instructions as insert, join, delete, define can change the length of a particular string. It is not desirable, when the length of the string is changed, to move any portion of the string in order to make it continuous. Therefore, a means of linking disjoint parts of a string will be provided.

This link will be composed of three characters, a special "go to" character indicating a link and a two character address locating the next word in the sequence. This group of characters will be placed at the end of the machine word which would normally precede the linked data. If this machine word contains information other than nulls in any of these three character positions, the characters are extracted and stored in an available word. The data to be linked is either stored in the following blank locations or another link is provided between the "prelink" word and the linked string. A link is placed at the end of this linked data back to the original string.

A special link symbol, ⑧ , permits the programmer to insert common data, which is stored once, into several strings. When this character is recognized by input control, this special character will be interpreted as an identifier in the sense that it forces a new machine word to be started with this character. In this case the character is stored in the first and sixth character positions of the new word, and the nulls are stored in the rest of the word. The name following the ⑧ character will be stored beginning in the next word. The first time that this data is used and the symbol interpreted as a link, the name identifying the common insert will be used to determine the address of the beginning and end of the data. These addresses will then be stored with the ⑧ link. The name will not have to be referred to in subsequent uses of the original data. At the end of the inserted string, will be placed a variable link. When this string is being used, the address of the next location in the calling sequence is placed in this variable link. Consequently, the link back to the calling sequence is provided at the time of reference.

Since a link indicates a discontinuity in a string of data, the location of a discontinuity must be available when the contents of an identifier plane are being searched and counted as described previously. Another special core plane--the Forward Discontinuity Plane--is provided. This will be accessed and searched in parallel with any of the identifier planes when a forward search is underway. A bit in this plane set to "1" indicates that a special circumstance applies to the corresponding machine word, and the machine word must be accessed before any search can continue. In general, a discontinuity will be indicated in that machine word and the search will be continued at the place indicated by the link address.

It is also possible to search backward from any point within a named string of data. However, only forward links are provided in the data. A push down store called the Reverse Push Down Store (RPDS) is used to store the reverse links generated as a forward search is conducted. Since one cannot access any piece of data beyond the extent of the name indicated, it is normally true that a forward search must precede any backward search and thus the reverse links are always provided in the RPDS. A Reverse Discontinuity Plane will also be provided to indicate the special circumstances that are pertinent to a backward search.

Let us say we are given the string A stored in memory as shown in Fig. 4. In this string there is a discontinuity at location (yy - 1). This is indicated in the forward and reverse discontinuity planes. The beginning of the strings are indicated in the reverse plane. The ends of the strings are indicated in the forward plane. It is required to find

$$A\ \textcircled{2}\uparrow\ 3\ \textcircled{1}\uparrow\ -\ 2$$

Once A has been found, the ② and the forward discontinuity plane are accessed and the count begun. At yy - 1 a discontinuity mark is reached. The machine word is accessed for the link address xx and the address (yy - 1) placed in the RPDS. The ② and forward discontinuity plane are accessed at xx and the search continued. At the end of the linked string, the same situation

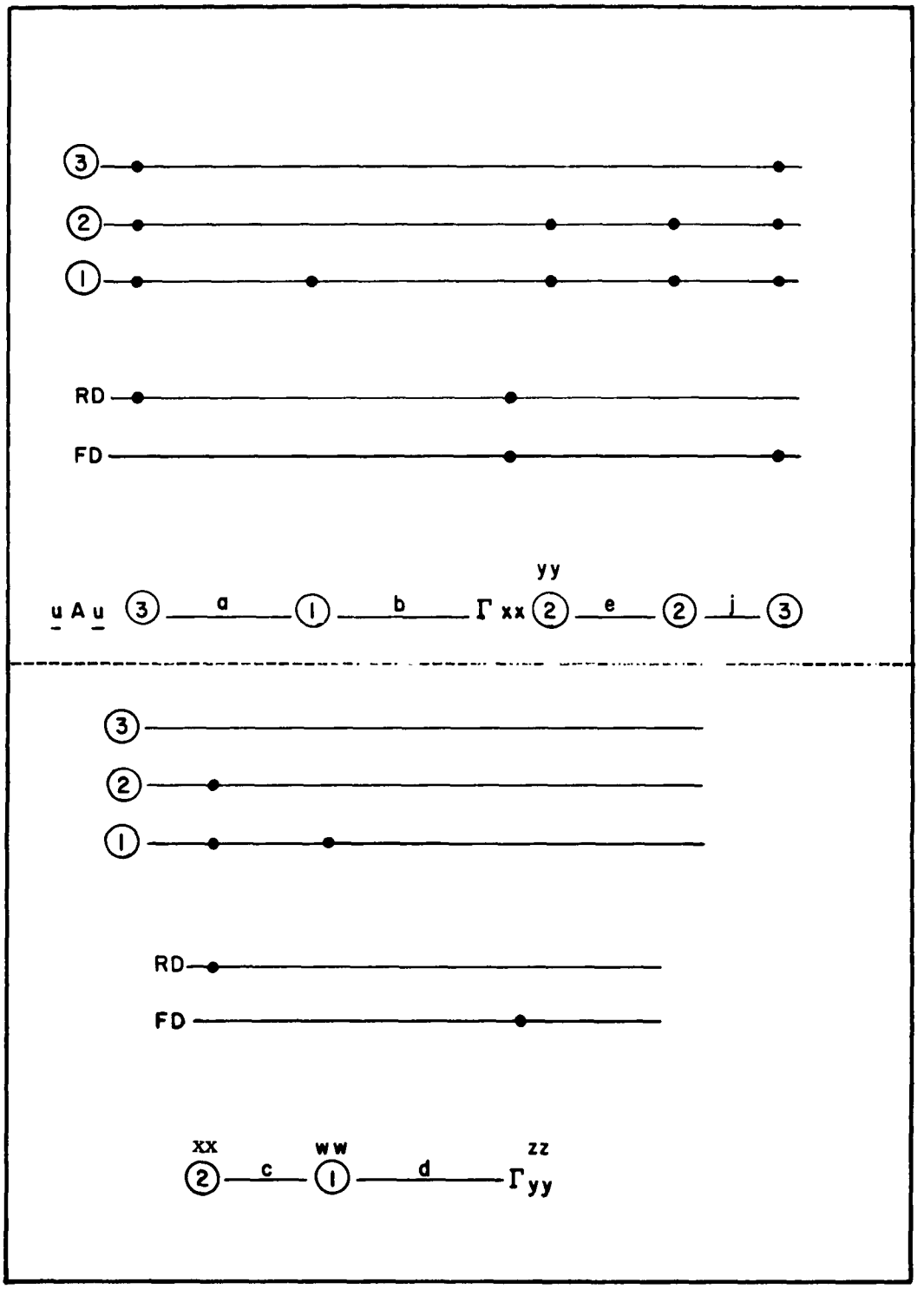


FIG. 4

is indicated. The address zz is placed in the RPDS and the search is resumed at location yy. The second ② mark beyond but including this point is the one indicated by A ②↑ 3 (or determined by a count of the ② marks as we have been searching). The search is then continued at the one level and in a reverse direction as indicated by the next adjective ①↑ - 2. At this time, the address zz is at the top of the RPDS and the address (yy - 1) below it. The ① and Reverse Discontinuity Plane are accessed and the search and count are continued backwards. Again at yy - 1, a discontinuity is searched, but this time the link address is obtained from the RPDS. Thus, the search is continued at zz. The next ① is the one desired so the search stops at this point and the RPDS is cleared since there are no further adjectives modifying the noun A.

If the noun phrase had been,

A ① ¢ 0 ①↑ - 2

and the current address was ww, a slightly different situation exists. After the name is located, the first adjective indicates the current symbol. This address is determined. The next adjective specifies a backward search from this point. The ① and Reverse Discontinuity planes are accessed and the search initiated. However, the search will stop at xx, and the word accessed. In this case, the controls will determine that this word is the beginning of a linked string of data but that the RPDS is empty. Consequently, a forward search out to this point must be accomplished so as to build up the RPDS. The backward search can continue as soon as this operation is complete.

ARITHMETIC OPERATIONS

One of the considerations used in the specification of this system was that all data must be considered to be variable field length. In this situation, numeric processing becomes more difficult when considered in the conventional manner.

A number of numeric operations are basically high to low order operations. These include input, output, division, comparison, and normalization. Considering the problems of implementing these operations together with those of the arithmetic operations, it was decided to process all data high order to low order. The effect of the extra complexity that may result in the process unit is hopefully offset by the increased efficiency of the system.

Basically, the addition of two numbers serial by character is quite similar when done either high order to low order or vice versa. In the latter case, two digits are added producing a sum and a carry or no-carry. The carry, no-carry information is stored for use in the next cycle--the addition of the next higher order pair of digits. In high to low order operation, the results of an addition are still a sum digit and a carry, no-carry indication. In this case, however, the sum digit is stored until the next cycle so that it can be modified by the carry, no-carry result of that operation.

However, there is a complication in high to low order processing. It can best be illustrated by an example. Consider the addition of the following pair of numbers:

$$\begin{array}{r} 24444\ n \\ +\ \underline{35555\ m} \end{array}$$

where n and m are any digits. Proceeding from left to right, the first sum is 5 with no carry. The 5 is placed in a buffer and the next pair of digits added. The result is a 9 with no carry. The 5 is now gated to a temporary store and the 9 into the buffer. The following cycle also produces a 9 and no carry. The previous sum is put into the temporary store and the nine into the buffer. This procedure continues until the last cycle. If m + n

produces a carry, this carry will not propagate through the string of nines formed and modify the first non-nine digit to the left of sequence in the result. Thus, the string of nines and the next higher order digit must be available to be changed by the carry as it is propagated.

One technique which accomplishes this is to avoid outputting any nine or digit which is in the next higher order position above a nine until it is known whether or not there is a carry to be propagated. This involves storing the next higher order digit (NHOD) and the ensuing consecutive nines until a non-nine digit is produced. At this point, the carry, no-carry information is consulted, and the NHOD and the nines are outputted after being incremented by one or zero.

When the first nine in a string is produced, the NHOD is already in a one digit temporary store awaiting carry, no-carry information from that cycle. To avoid outputting anything until a possible carry propagation can occur, the nine output is counter, and the NHOD output is prohibited. Since only nines need to be stored in this manner, a simple counter is sufficient to keep track of consecutive nines.

Finally when the output of the process unit is a non-nine digit, the NHOD is outputted after being modified by the carry, no-carry information of this cycle. The counter is then decremented by one and a nine outputted. This digit is also modified by the carry, no-carry information. This procedure is continued until the counter is cleared. The non-nine digit is placed in the temporary storage, and the processing continues on the next pair of digits.

Basically, then, the procedure is to count the nines as they are produced to inhibit all process unit output until a non-nine is detected, and then to correct for the carry as the digits are outputted.

High to low order subtraction offers no additional complexity over that present with addition. Subtraction may be accomplished by complementing the subtrahend and adding the result to the minuend. If the nines complement

is used, the addition proceeds with exactly the same rules for nines handling and carry, no-carry modification as was used for addition. If the difference is in non-complement form, a 1 must be added to the lowest order digit to take care of the end around carry. If the 10's complement is taken, zeros must be counted in the same manner as the nines were in addition, and the output must be decremented by zero or one to effect the carry, no-carry propagation.

Complemented answers will be avoided by always subtracting the smaller magnitude number from the larger. If this condition does not exist initially, the roles of the minuend and the subtrahend can be interchanged by changing the true-complement sense of both operands and changing the sign of the result.

The need for a reversal can be established only after determining which operand is larger, which in turn depends upon the magnitudes of the highest order not equal pair of digits in the operands. The relative magnitude of the first pair of unequal digits sent to the process unit will determine if a reversal is necessary when using high to low order processing. The results obtained from higher order pairs of equal digits (if any) will be the same whether a reversal occurs or not. Thus, if a reversal is necessary, only the digit cycle in which the need for reversal is detected must be re-run. With or without reversal, the remainder of the subtraction is completely normal. In either case, only one pass through the process unit is necessary. This is a distinct advantage of high to low order processing.

Multiplication and division offer no additional problems. Multiplication may be done either by repetitive addition or by multiplication of multiplier and multiplicand digits in an N-tupler and by addition of the result to the partial product. In either case, the addition will be performed as described above. If N-tupling is used, the multiplication itself is insensitive to the direction of processing. Similarly, division may be done by repeated

subtraction, accomplished as described above, or by the estimation of a quotient digit, multiplication by the divisor digits, and subtraction of the result from the dividend. The subtractions may be performed as previously described, and the multiplication is insensitive to the direction of operation.

The numeric fields to be processed normally are in normal form for an arithmetic operation. Normal form in the system is considered to be

exp + - - + - - - - - -

or exponent character, sign, two digit exponent, sign, implied radix point and mantissa. Usually numeric fields will be normalized during the input operation. A numeric field is a field containing only numeric characters (this includes exponent character, radix point and signs). However, a radix point must occur if the field is to be stored in normal form.

Addition and subtraction can be done on integers without forcing the operands to be in normal form. This ability is included in the system to make indexing as efficient as possible. However, both operands must be in this form, or both will be normalized before the operation begins. The operands must be normalized for multiplication and division.

A two accumulator system is used with the process unit. High to low order multiplication and division are such that the additional controls necessary for use with a single accumulator require enough extra hardware and slow the operation enough that a second accumulator is justified. (This second accumulator also speeds up some of the other automatic housekeeping and data handling chores in the system and could be justified nearly on this account alone.)

EXTENSION OF THE LANGUAGE

The most important property of the machine language is its ability to be expanded. This ability is accomplished through a means similar to the method used to construct subroutines in present day computers. However, once an operation is defined, it can be used in the same way any other operation in the language is used. There are no special rules for the use of these defined operations.

In the language some verbs, modifiers, and certain nouns have been defined and each assigned a special symbol. However, any combination of general characters may be used not only as a name for some data but also for some sequence of instructions. The only requirement for a particular combination of letters to be used as the name of an operation is that this combination not be used with any other meaning either as an operation or as a noun; that is, it must have a unique definition.

New operations may be defined using the previously defined operations and nouns. In addition, the noun "op n" is available for use in communication between the definition and the use. This noun will indicate the nouns included in the calling sequence. If any one noun in the calling sequence is required--the nth noun, for example--then this is indicated by writing op n. As an example, let us define some verb "triple add" as follows:

v triple add v ③ op 1 + op 2 + op 3 ③

The instruction

③ (triple add, A, B, C) →D 3,

will place the sum of A, B, and C into location D.

The simple rule for the formation of a defined operation is that the last sentence of its definition must be able to replace the calling expression without violating any syntactic rules. In this replacement, the first and last identifiers in the last sentence of the definition are neglected. In the example above, the rule was obeyed since ③, (A + B + C)→D 3, is a complete sentence.

In defining a new operation, it is permissible to use other defined operations. An operand noun may, in turn, refer to another operand noun. For example, let us define the operation sq $\overset{s}{p}$ rt as follows:

```
v sq p rt v ④ op 1 → Xi
            ③ (Xi → Xj
            ② .5 * (Xi + op 1/ Xi) → it)
              until (it - Xj) abs < ϵ
            ③ Xi
            ④
```

Further, we can define the operation sq $\overset{s}{p}$ root using sq $\overset{s}{p}$ rt as follows:

```
v sq p root v ③ (sq p rt, op 1) → op 2  3)
```

Thus, the instruction ③ (sq $\overset{s}{p}$ root, A, B) ③ will obtain the square root of A and place this in B.

There is no limit to the number of operands nor to the number of levels in a defined operation.

A Subroutine Push Down Store (SPDS) will be used to store the addresses to which subroutines must return when they are completed. Whenever a name in an instruction is found to be a defined operation (surrounded by v marks), the address of the name location in the instruction is placed in the SPDS and control is transferred to the indicated subroutine. When the subroutine is finished, control is returned to the point indicated by the top address in the SPDS. Because there is no limit on the size of the SPDS (other than the size of main memory), there is no limit on the number of levels of subroutines. A defined operation may use a defined operation, may use a defined operation, etc. These addresses in the SPDS may also be used to obtain the appropriate parameter whenever an "operand" noun is mentioned. A special table, however, will be used for this purpose in order to make the interpretation of subroutines more efficient. Whenever a defined operation is found, the absolute addresses of its parameters are

placed in a table. When a subroutine is first encountered in the original program, the first parameter is placed in location 11 (corresponding to the first parameter, first level), the second in 21, etc. If another subroutine is reached within the first subroutine, the absolute address of its parameters are placed in locations 12, 22, 32, etc. If an operand noun, say operand m, is reached in subroutines at level n, then the absolute address corresponding to this noun can immediately be found at location m, n in the table. The o, n location at each level is the absolute address of a temporary result string used at the particular level and addressed at each level by the noun temp. The size of this special table will limit the number of parameters and number of levels at which this interpretation can be efficiently performed. The table used in this system will allow 32 levels and 9 parameters at each level to be evaluated efficiently.

INSTRUCTION SEQUENCING

As discussed in previous sections, instructions in the system language are sentences within some defined operation. If the defined operation is at level 0 (the main program in the usual sense), the system will step through this operation, sentence by sentence. Each sentence in this case must be syntactically independent and complete. The execution of the sentences will continue to the end of the sequence in which it is started unless a programmed transfer to another defined or named operation is encountered. This is the usual transfer encountered in a branch instruction in a conventional system. If another defined operation is used in a noun phrase in a sentence, control will be transferred to that operation and returned to the sentence as soon as the operation has been completed and the resultant noun phrase evaluated. In this case, the detection of the completion of the execution of the operation and the return are automatically determined and need not be programmed explicitly.

The proper sequence of the operations is accomplished through the use of a push down store. This store will serve as a means of reversing the sequence of any part of an instruction when this is necessary for proper evaluation. This storage will also save the starting point of any recursive loop in the program. The effect of each operator and operand on its sequencing depends on the verb itself and its context. The store, itself, will consist of a fixed sequence of memory locations. Since these will be sequential, no linkage is necessary. However, if at any time more storage is required, as many available machine words as necessary will be automatically added to the push down store. At the beginning of each overflow word, will be a link to the location of the previous machine word in the push down store.

This push down store will have the usual purpose of a "last in-first out" storage which, in effect, reverses the sequence of items from the order in which they are read in.

As an instruction is being interpreted, several quantities are accumulated and may need to be retained. As an example, the extent of an operand (beginning and end addresses of its location in the storage) is determined when that operand is mentioned in the instruction. This information is normally stored in address registers associated with the subject and object register storage. If the operation using this operand cannot be executed at this time, the operator and operand specifications must be placed in the push down store. Some special characters such as "(" or " ③ " must have their machine word and character addresses stored as well as the characters themselves. Consequently, each machine word in the push down store will contain only one entry; that is, operand, operator, or special character.

As the instruction is executed, temporary or partial results occur and must be stored. These results will be placed in a temporary store which is constructed similarly to the push down store. The words of this store are sequential and can be added to from main memory. This store is useful in that it eliminates machine map, search, and delete operations for these resultant operands. The extent of the operand is noted and retained when the items are placed in this store. The items are removed from the store and the space made available automatically as the sentence execution proceeds.

At the end of a level 0 sentence, both the instruction push down store and the temporary store are cleared.

AUTOMATIC INPUT AND OUTPUT

Since the programmer has no control over where data is stored in a proposed system and how much of the store a given block of data occupies, the machine must have control over some input and output operations. Part of this control is accomplished through a register in which is stored the current size of the availability list. This availability list is a linked list containing all the available locations in the store. The contents of the register are used to determine when additional space is needed in storage or when there is sufficient space to allow another block of data or instructions to be loaded into memory. Thus, when the main store is approaching some fraction of full capacity, it will output data. When the memory is relatively empty, it will input data. Once data has been in the machine, the link address associated with the names will always permit the machine to find it and return it to the main store if necessary. The programmer need only provide the machine with an input list indicating where named blocks are located and a priority, and a "scratch paper" list indicating where data should go temporarily and a priority. The final output of results will occur as programmed.

The input list will have a sturcture similar to the data itself. It will be a sentence whose symbols contain the name of the string of data to be inputted and the indication of where in external storage the data is located. Those strings of data that have the same priority are grouped into phrases. Thus, if two strings of data will be required at a given time, they are given equal priorities by placing them in the same phrase. The priority of the phrase is indicated by the order in which it occurs in the input sentence. Each symbol in the input sentence is of the form:

Alpha ③ unit 1

where Alpha is the name of some data string and unit 1 is the name of the external storage. The identifier mention is used to indicate the level of the named block to be input if the whole block is not to be moved at one time.

A typical input sentence might be:

③ Alpha ③ unit 1
1 Beta ④ unit 1
2 Gamma 3 unit 2
2 Matrix A 2 unit 2
1 Matrix B 2 unit 1
2 Dictionary 4 unit 1
③

The input list will be used when the memory load is some fraction of full capacity. The automatic input will continue until the memory reaches some fraction of capacity above which the input will cease at the next permissible point. The particular values have been chosen to be . 2 and . 6, respectively.

Data can also be inputted into the machine by instruction. First, if a named string is defined as being the contents of an external store, the input of the contents must occur. Next, if the operand specified in an instruction is not in the machine but can be located by the system, enough data must be scanned and sufficient data inputted to locate the operand. During automatic input, the normal program execution continues and is interrupted only by the priorities assigned to machine operations for use of the memory. The define instruction which causes an input also includes the point beyond which the normal program cannot proceed until the input is completed. This permits simultaneous input and compute, with as little wasted machine time as possible. Compute must stop while an operand is being located.

If part of a string of data is inputted, a link will be placed at the end of the stored data to the external storage where the rest of the data is located. The exact form of the address will depend upon the organization of the data in the external store.

The input list always has a current address associated with it. This address indicates the data which is to be inputted next. When the data indicated in an input phrase has been completely inputted, the "current" indication is moved to the next phrase on the list. Thus, when an input is called for, the current phrase will contain the names and locations of one or more data strings. These strings may be:

1. Completely in the external storage and none yet inputted into the main internal store.
2. All partly inputted into the main store.
3. Some completely inputted into the main store but some only partly inputted.

The scratchpad list is used to specify temporary external storage for data and intermediate results during a computation. This list has the same characteristics as the input list with the exception that entries are placed on this list by programmed statements. This list is used in much the same way as the input list in that there is some fraction of full memory capacity above which output will occur and some fraction of full capacity below which this output will cease. Provisions are made to include the proper linking so that all parts of the data, whether in internal or external stores, are available to the system.

The final output of data must be programmed. When an external store is used as the object of a define verb, a point in the program beyond which the program execution cannot proceed is also included to permit simultaneous output and compute. If both programmed input and output are happening at any given time, the machine will stop at the first specified "wait" point until both operations have been completed.

CONCLUSION

The sections of this report have described a machine language and organization for an experimental digital data system. This system has been developed and designed in an attempt to provide the advanced or experimental programmer with a tool which can be used to solve many of his problems more easily and efficiently. It is expected that experience with this system will point out areas in which more work needs to be done and others in which the problem has been overestimated. It is anticipated that from such experience more sophisticated and perhaps simpler ways of solving the same problems that have been tackled here will be found. However, it is felt that the first step had to be taken no matter how faltering or short it may be. The actual evaluation of this step now awaits the finish of the logical design and the subsequent construction of this system.

Electronics Research Directorate, Air Force Cambridge Research Laboratories, Bedford, Mass.
Rpt. No. AFCRL-62-964. ADAM - A PROBLEM ORIENTED SYMBOL PROCESSOR. Scientific rpt. no. 1, 20 Dec. 62, 36 p., incl., illus. IBM Research Report No. RC-840.
Unclassified Report

A study of the general area of problem solving with a digital computer revealed characteristics of data that are essentially ignored or suppressed in conventional systems. In an attempt to increase the capability and flexibility of a digital system a new high-level language has been defined which utilizes these data characteristics. A machine organization which implements this language as a machine language and yet imposes no restrictions on the use of the language has been proposed. The system will have the following characteristics as a result of this organization:
1. Complete symbolic addressing on variable field length

1. Computer Logic
2. Data Processing Systems
3. Programming Languages
4. Digital Computers
I. AFCRL Project 4641, Task 464105
II. Contract AF19(628)-1621
III. Thomas J. Watson Research Center, IBM, Yorktown Heights, New York
IV. A. P. Mullery, R. F. Schauer, R. Rice
V. In ASTIA collection